

NEWTON'S METHOD IN PRACTICE II: THE ITERATED REFINEMENT NEWTON METHOD AND NEAR-OPTIMAL COMPLEXITY FOR FINDING ALL ROOTS OF SOME POLYNOMIALS OF VERY LARGE DEGREES

DIERK SCHLEICHER AND ROBIN STOLL

ABSTRACT. We present a practical implementation based on Newton's method to find all roots of several families of complex polynomials of degrees up to 134 million so that the observed complexity to find all roots is between $O(d \ln d)$ and $O(d \ln^3 d)$ (measuring complexity in terms of number of Newton iterations or computing time). All computations were performed successfully on standard personal computers made in 2010/2011, using only a single processor core.

CONTENTS

1. Introduction	1
2. Background on Newton dynamics	3
3. The Iterated Refinement Newton Method	4
4. The polynomials investigated and the results of experiments	7
4.1. Centers of hyperbolic components of the Mandelbrot set	7
4.2. Periodic points of $z^2 + 2$	10
4.3. Periodic points of $z^2 + i$	12
4.4. Computing time per iteration	13
5. Guaranteeing that all roots are found	14
5.1. Guaranteeing that all roots are found	14
5.2. Newton identities	15
5.3. Newton identities in practice and precision estimates	16
5.4. Conclusion	19
6. Finding missing roots	19
6.1. Implicit deflation	20
6.2. Ehrlich-Aberth iteration	20
6.3. Newton identities	20
6.4. Newton identities in practice	22
References	25

1. INTRODUCTION

It has been known since Gauss that every complex polynomial p in a single complex variable splits into linear factors, and since Ruffini and Abel that there is no method based on iterated n -th roots to find these factors algebraically in

general. Therefore, numerical approximation methods are required to find the roots of polynomials.

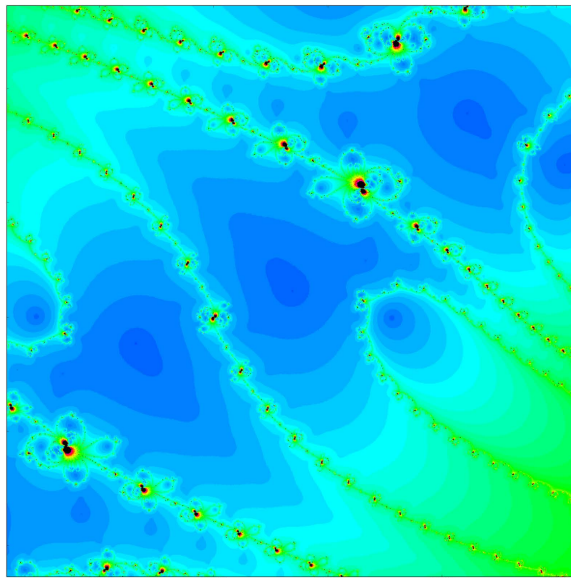


FIGURE 1. The Newton dynamics of a polynomial of degree $2^{27} > 10^8$ (detail). The colors illustrate the number of iterations until a root is found to high precision; adjacent color contour lines show a single iteration. The scale is between 50 iterations (red) and 0 iterations (dark blue); bright green stands for 25 iterations. Compare with Figure 18. Picture by Marvin Randig.

Surprisingly, even today it is not clear how to find all roots most efficiently. There is a theoretically best possible algorithm due to Pan [P2], but it is not applicable in practice. Perhaps the most efficient practical implementation is `MPSolve 3.0` by Bini and Robol [BR14] based on the Ehrlich-Aberth iteration, but it is still lacking good theory and especially a proof that it is generally convergent.

Newton’s method is very good at finding roots locally, i.e. once good approximations are known, but it has a reputation of being difficult to predict globally, for instance because of the “chaotic” nature of the iteration, and because of the possibility of having open sets of starting points that converge to attracting periodic orbits, rather than to roots.

The situation has improved substantially in recent times. Indeed, a pleasant feature of Newton’s method is that one does not have to choose between theory and practical implementations: meanwhile it has good theory (see [HSS, BLS] on where to start the iteration, and [S4] with refinements in [BAS] on estimates on its complexity that are near-optimal under certain assumptions), and it can be implemented quite successfully in practice for polynomials of very high degrees. The latter aspect is the key theme of the present manuscript.

In [SSt1], we showed that Newton’s method can be used to find all roots of various complex polynomials efficiently, even up to degrees greater than one million

(where the polynomials were selected only on the criterion that they could be evaluated efficiently). In this note, we demonstrate that the computational complexity (measured in computing time) can be near-optimal for a wide range of degrees: all roots can be found in computing time between $O(d \ln d)$ and $O(d \ln^3 d)$, and in a number of Newton iterations between $O(d \ln d)$ and $O(d \ln^2 d)$, with small constants, so that all roots can be found for degrees up to 2^{27} (greater than 130 million), and even higher degrees seem feasible.

We present an algorithm based on Newton’s method that turns out to find all roots, the families of polynomials studied, and we describe the outcome of computer experiments. We also explain how to verify that indeed all roots of our polynomials have been found.

This note should be seen as a continuation and improvement on [SSt1], not as a survey on root finding for polynomials. There are interesting references on the latter, in particular the survey articles by McNamee [McN1, McN2], as well as studies by Pan [P1, P2], Renegar [R], and especially MPSolve 3.0 from [BR14] and Eigensolve from [F].

Acknowledgements. We are very pleased to be able to report support, encouragement, and helpful suggestions from numerous friends and colleagues, especially Dario Bini, Marcel Oliver, Victor Pan, Marvin Randig, Simon Schmitt, and Michael Stoll. This research was supported in part by the Advanced Grant “Hologram” of the European Research Council (ERC).

2. BACKGROUND ON NEWTON DYNAMICS

There are a number of theoretical results available on the global dynamics of Newton’s method $N_p = \text{id} - p/p'$ for a given polynomial p . All require that a disk is known that contains all roots of p . After appropriate rescaling, we may assume that all roots of p are contained in \mathbb{D} .

We showed in [HSS] that there is a universal set of $1.1d \ln^2 d$ starting points, placed on $O(\ln d)$ concentric circles containing $O(d \ln d)$ points each, so that Newton’s method started at these points will find all roots of all polynomials of degree d . This was improved in [BLS] to a probabilistic set of starting points containing only $O(d(\ln \ln d)^2)$ starting points that find all roots with arbitrarily high probability. Our earlier experiments in [SSt1] started with $4d$ or $8d$ equidistributed points on a single circle; in many cases, $4d$ points were sufficient, but always $8d$ points were enough: this shows that in practice $O(d)$ starting points seem to suffice, even though this is not supported by theory (and there may well be special polynomials that require more starting points). Of course, any additional factors like $\ln^2 d$ are hard to detect numerically, even when the degree ranges up to 2^{27} .

Theoretical upper bounds on the expected number of required Newton iterations were given in [S4, BAS]; these are, up to polylogarithmic factors, on the order of d^3 or even d^2 .

However, these previous methods all require at least $O(d^2)$ iterations in order to find all roots: the methods rely on controlled Newton dynamics away from \mathbb{D} , so all starting points z are uniformly bounded away from the disk: $|z| > r > 1$ for a uniform r . However, on this domain we essentially have $N_p(z) \approx dz/(d-1)$ [HSS, Lemma 3], so each orbit needs at least $O(d \ln r)$ iterations until it even enters \mathbb{D} , and since at least d orbits are required to find d roots, we obtain a lower bound for

the complexity of $O(d^2)$ Newton iterations. The upper bounds in [BAS] are thus best possible, again up to polylogarithmic factors.

One may try to remedy this problem by starting the iteration on a circle of radius, say, $1 + 1/d$. Such an approach comes with various problems. One is that we are losing the theory to guarantee that all roots will be found. Another one is that this will be helpful only if a very tight bound for the smallest disk containing all roots is known, and only if most roots are near the boundary of that disk. This is not the case for instance for all polynomials considered by us (but it is true if the coefficients of the polynomial are independently randomly distributed [ET, Ar]).

3. THE ITERATED REFINEMENT NEWTON METHOD

One key observation is that the lower bound of the number of iterations comes from the iterations outside of \mathbb{D} , that is before the interesting dynamics even starts, and where all starting points are on very similar “parallel” orbits — a necessary consequence of the fact that we start the Newton dynamics at points with controlled dynamics. Typical initial orbits of the Newton iteration are shown in Figure 2. The approach we take in the experiments described in this paper is that on domains where orbits are “parallel”, fewer orbits are required, as indicated in Figure 3.

More precisely, we start the Newton dynamics with a fixed number of, say, 64 Newton orbits on a circle away from \mathbb{D} where we have good control. Along each triple of adjacent orbits, we compare the shapes of the triangles formed by the three points on adjacent orbits. As long as this shape remains nearly constant, so that the three orbits form nearly similar triangles, we keep iterating these three orbits, assuming that they represent similar dynamics for all orbits that might run between these three “representative” orbits. Once the triangles start to deform, we split up these orbits by inserting an additional orbit half way between each of the two pairs of adjacent orbits; see Figure 3.

Of course, this requires a heuristic refinement threshold parameter that describes the threshold of deformation after which parallel orbits are refined. We use the cross-ratio between three adjacent orbits with respect to ∞ : that is,

$$(1) \quad t_{n,i} = \frac{z_{n,i-1} - z_{n,i}}{z_{n,i+1} - z_{n,i}}$$

where the index n counts iterations and the index i counts the circular order of the orbits (with due modifications at the end to turn the linear order of natural numbers into a circular order). That is, without refinement, we have $z_{n+1,i} = N_p(z_{n,i})$; and when refinements occur, these are inserted into the circular order (and the indices i are shifted accordingly).

Our refinement condition is essentially based on $\alpha_{n,i} := |\ln(t_{n,i}/t_{n_0,i})|$, where $n_0 \leq n$ is the most recent iteration when a refinement occurred for this orbit (properly accounting for index shifts because of refinements that might have occurred for other orbits).

The algorithm is determined by a number of parameters that we determined heuristically:

- the number N_0 of initial orbits (we usually use $N_0 = 64$, but this should not make much of a difference);
- the maximal number of iterations a particular orbit can run for (we use $10d$); more important than this number is a good detection of periodic cycles (if a

positive fraction of orbits runs to the maximal number of allowed iterations, we end up at complexity $O(d^2)$;

- the maximal number of refinement generations (we use $\log_2(4d/N_0)$; that means that if all refinements have taken place, we end up with $4d$ orbits);
- the refinement threshold R : that is, the maximal value of $\alpha_{n,i}$ that is allowed before further refinement occurs. Most of the time, we use the value of $R = 0.05$, but for some experiments we had to increase sensitivity to $R = 0.0005$;
- and a stopping criterion when success is declared that some orbit found a root: we stop when $|N_p(z) - z| = |p(z)/p'(z)| < \varepsilon_{\text{stop}}$; in practice, we use $\varepsilon_{\text{stop}} = 10^{-15}$ or sometimes 10^{-16} (while we used the `long double` data type with numerical precision of about 18 relevant digits).

Finally, one needs a post-processing step: many roots will be found by several different orbits, and we have to make sure these roots are accounted for only once.

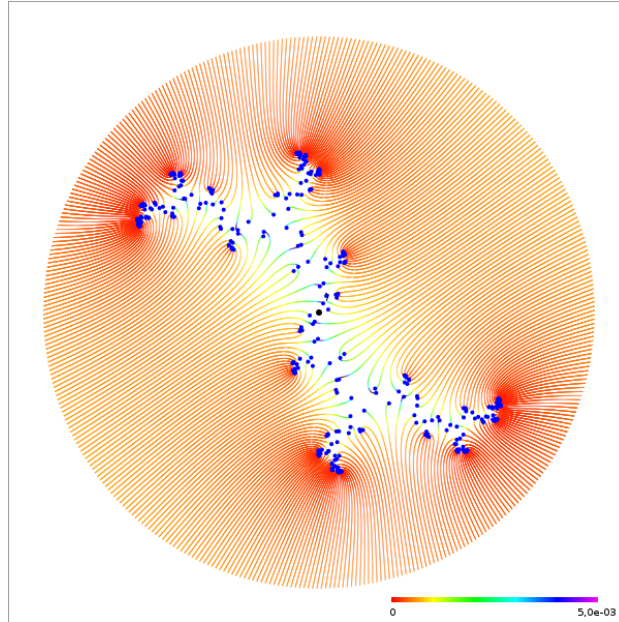


FIGURE 2. The Newton iteration for 400 starting points on a circle of radius $r = 2$ (here for a polynomial of degree 4096, so we do not have enough starting points to find all roots; the polynomial shown here describes periodic points of period dividing 12 of $z \mapsto z^2 + i$). The apparent lines connect orbits under the Newton dynamics; colors indicate the number of iterations until an approximate root is found. The behavior of the iterations outside of the disk containing all roots is very parallel and “wasteful”, but required to carry over the control from the circle of starting points to the interesting dynamics on \mathbb{D} . Observe that even if we had a very precise bound on the smallest disk containing all roots, this would not help much as most roots are away from the boundary of this disk.

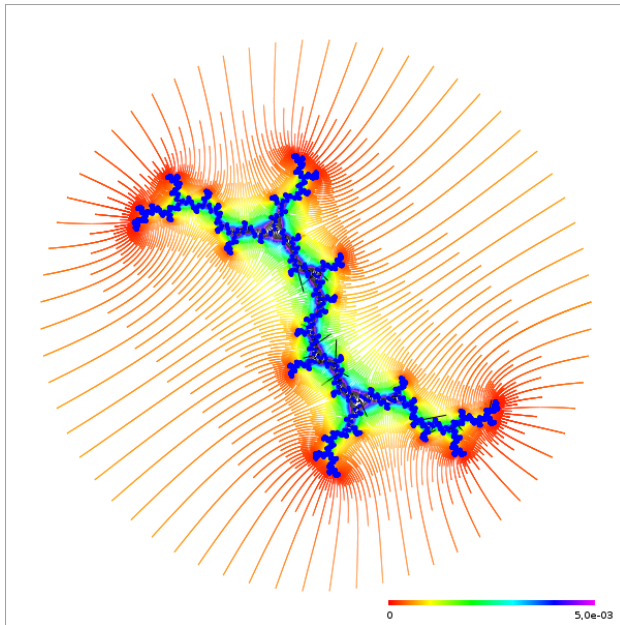


FIGURE 3. The Iterated Refinement Newton Algorithm: we start with 64 points and refine only when adjacent orbits stop moving in parallel. As a result, most Newton iterations are spent near the roots, and far fewer iterations are required. The polynomial is the same as in Figure 2 (degree 4096).

Here we use very simple-minded heuristics and declare that two orbits found different roots if they terminate at distance greater than $\varepsilon_{\text{root}}$; here we use $\varepsilon_{\text{root}} = 10^{-14}$ (sometimes $\varepsilon_{\text{root}} = 10^{-15}$). Of course, this requires an efficient way to sort the roots by mutual proximity (when all roots are found by an algorithm that is linear up to log-factors, then for naive sorting procedures it is easy to spend more time on sorting than on root finding).

Of course all quantities are heuristic, and especially the choice of stopping criterion and post-processing (distinguishing different roots found) have obvious difficulties in the presence of high degrees or near-multiple roots. Our point is not that this is a strong part of the algorithm, but that even with such a simple-minded approach all roots can be found for all the polynomials investigated, and for extremely large degrees. An improved approach for more “challenging” families of polynomials, especially with near-multiple roots, is discussed in [SSch].

In Section 5 we describe several methods that establish a posteriori that all roots have been found with high accuracy.

4. THE POLYNOMIALS INVESTIGATED AND THE RESULTS OF EXPERIMENTS

We investigate (a subset of) the same polynomials we already discussed in [SSt1]. We will briefly introduce the relevant families of polynomials and then present the results of the experiments. These polynomials have been chosen solely for the purpose of fast evaluation (in terms of recursion): our focus is on root finding not efficient polynomial evaluation, so we chose polynomials where evaluation is easy. Therefore, comparing our experiments with other experiments on different polynomials in terms of computing time yields an unfair bias in our favor, but comparisons in terms of the required number of Newton iterations should be meaningful.

4.1. Centers of hyperbolic components of the Mandelbrot set. These polynomials in a variable c are defined by recursion $p_0 = 0$, $p_{n+1} = p_n^2 + c$, so we have $p_1(c) = c$, $p_2(c) = c^2 + c$, $p_3(c) = (c^2 + c)^2 + c$ etc., so that p_n has degree 2^{n-1} . Roots of p_n are those parameters c for which the iteration $z_0 = 0$, $z_{n+1} = z_n^2 + c$ is periodic with period (dividing) n ; this implies that p_k divides p_n when k divides n (using the fact that all roots of all p_n are simple; compare [DH, Section 19]).

Roots of p_n are known as *centers of hyperbolic components of the Mandelbrot set* of period n (except those that are already roots of p_k with $k|n$). We found all roots for $n = 25$ (i.e. degree 2^{24} , greater than 16 million) in less than 160 hours (on a standard PC using a single core). The detailed results are tabulated in Figure 4. In particular, we plot the complexity in terms of required Newton iterations in Figure 5: the diagram clearly shows that the complexity in terms of Newton iterations scales better than $200 d \ln^2 d$. The complexity in terms of computing time is shown in Figure 6; it seems to scale better than $10 d \ln^3 d$. This is of course related to the fact that our degree d polynomials can be evaluated in $\ln d$ operations. We realize that this is an untypical advantage of our polynomials. However, for polynomials given in coefficient form, there are fast methods of parallel evaluation at many points that should compensate for much of the complexity gain [MB], [AHU, Section 8.5] (these methods are efficient only when the number of evaluation points is comparable to the degree of the polynomial, which is the case for the Newton algorithm because eventually all roots have to be found by their own Newton orbit).

In order to find all roots, a significantly more sensitive refinement threshold of $R = 0.0005$ was used (compared to $R = 0.05$ used in the other experiments).

We should point out that Newton's method occasionally encounters attracting periodic orbits of periods greater than one. The total number of orbits that were detected to converge to attracting cycles of periods 2 or more is shown in Column J of Figure 4; for large d , this number seems to stabilize near $0.0037 d$ (Column K and Figure 7). While the total number of such orbits is relatively small, the orbits do not satisfy the easy-to-detect termination condition that a root is found, so the iteration times can be large. Hence, if such orbits are not caught efficiently, they will consume a lot of computing time. It turns out that for the Mandelbrot center polynomials, far more attracting cycles were found than in most other experiments. For comparison, for the other families of polynomials investigated here (periodic points of $z^2 + 2$ and $z^2 + i$), for degree 2^{21} no attracting cycles were found at all, and for degree 2^{27} less than one hundred orbits converged to attracting cycles.

A	B	C	D	E	F	G	H	I	J	K
period n	degree $d=2^{n-1}$	seconds	hours	iterations/d	iter/d($\ln d$)	iter/d($\ln d$) ²	$\mu s/d(\ln d)^2$	$\mu s/d(\ln d)^3$	attracting cycles	attr cycles * 1000/d
10	512	1	0.00	5073	813.20	130.36	50.19	8.05	6	11.72
11	1024	2	0.00	7242	1044.80	150.73	40.65	5.86	0	0.00
12	2048	7	0.00	9544	1251.73	164.17	58.79	7.71	20	9.77
13	4096	19	0.01	12202	1466.98	176.37	67.05	8.06	15	3.66
14	8192	52	0.01	14910	1654.66	183.63	78.18	8.68	45	5.49
15	16384	134	0.04	17790	1833.25	188.92	86.85	8.95	56	3.42
16	32768	334	0.09	20687	1989.67	191.37	94.29	9.07	139	4.24
17	65536	813	0.23	23662	2133.57	192.38	100.86	9.09	223	3.40
18	131072	1914	0.53	26421	2242.20	190.28	105.17	8.93	472	3.60
19	262144	4489	1.25	29669	2377.96	190.59	110.01	8.82	884	3.37
20	524288	10312	2.86	32287	2451.59	186.15	113.40	8.61	1911	3.64
21	1048576	23438	6.51	35374	2551.69	184.07	116.31	8.39	3901	3.72
22	2097152	52443	14.57	38351	2634.70	181.00	118.02	8.11	7944	3.79
23	4194304	117204	32.56	41348	2711.48	177.81	120.17	7.88	15543	3.71
24	8388608	283134	78.65	43951	2756.86	172.93	132.80	8.33	31419	3.75
25	16777216	566157	157.27	47232	2839.22	170.67	121.94	7.33	62298	3.71

FIGURE 4. Results for finding all centers of hyperbolic components of the Mandelbrot set for period $n \leq 25$, i.e. for degrees up to $2^{24} > 1.6 \cdot 10^7$. The first four columns specify period n and degree $d = 2^{n-1}$, as well as total computing time in seconds and hours. Column E gives the required number of Newton iterations (divided by d), while columns F and G scale this with respect to $\ln d$ and $\ln^2 d$. Columns H and I specify computing time (in microseconds) divided by $d \ln^2 d$ and $d \ln^3 d$. Column J shows the number of orbits that converge to attracting cycles, and the final column K shows the same number times $1000/d$. Refinement threshold $R = 0.0005$, all roots were found.

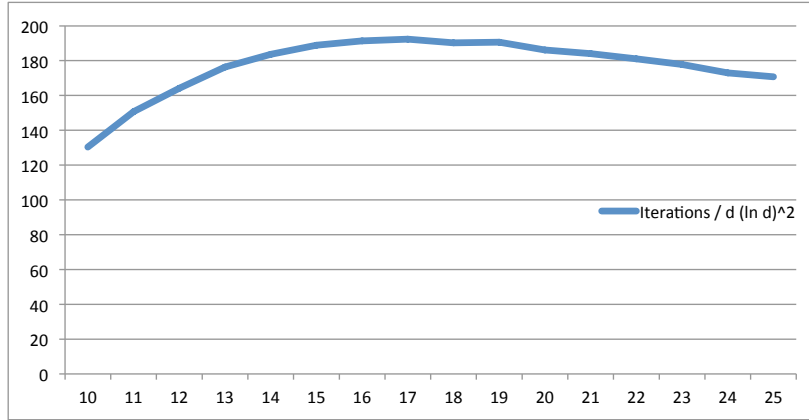


FIGURE 5. Number of Newton iterations, divided by $d \ln^2 d$ (column G) vs. period. Evidently the number of required iterations scales better than $200 d \ln^2 d$.

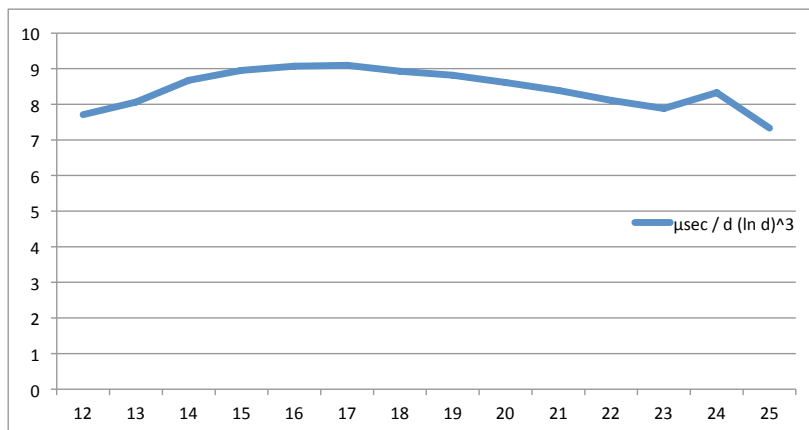


FIGURE 6. Complexity of finding all centers of hyperbolic components of the Mandelbrot set, in terms of computing time, in microseconds divided by $d \ln^3 d$ (column I). Evidently, the time complexity scales better than $10 d \ln^3 d$ (in μsec).

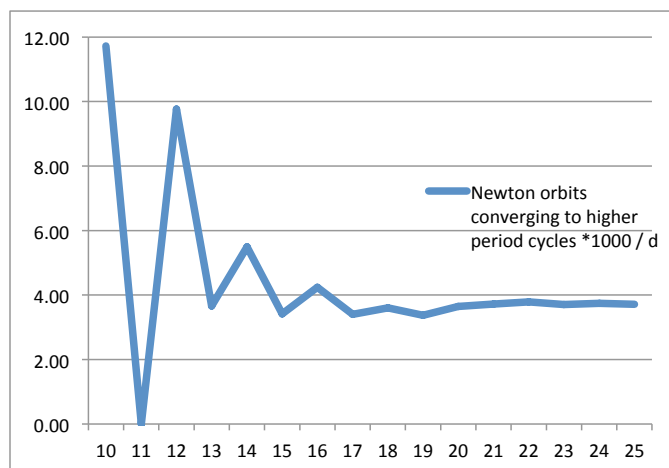


FIGURE 7. Number of Newton orbits converging to attracting cycles of higher period, times $1000/d$: for large d , this number seems to stabilize near $0.0037 d$.

4.2. Periodic points of $z^2 + 2$. For the quadratic polynomial $p_2(z) = z^2 + 2$, a periodic point of period n is a root of $p_2^{\circ n}(z) - z$, where $p_2^{\circ n}$ stands for the n -th iterate of p_2 . There are exactly 2^n periodic points of period n . We found all periodic points of period $n \leq 27$, i.e. for degree up to 134 million, using a refinement threshold $R = 0.05$. The largest degree polynomial took about 89 hours. The results are given in detail in Figure 8. The complexity in terms of required Newton iterations is approximately $2.67d \ln^2 d$ (see Figure 9). The computing time (shown in Figure 10) is in the order of $0.05d \ln^3 d$ for degrees d up to 134 million. However, it oscillates significantly by almost a factor of 2. Since we could not detect a clear reason for this oscillation, we ran the experiment twice. The outcome is very similar, except for large periods. These oscillations have no impact on the overall results of our experiments (but should be investigated in future experiments)

Note that this experiment would be outright impossible when expressing $p_2^{\circ n}$ in coefficient form: the constant coefficient alone would have magnitude greater than 2^{2^n} , so for $n = 27$ greater than $2^{134\,000\,000}$. Just storing this number would require about 15 megabytes per coefficient (note that good relative precision is not sufficient because all the large terms are subtracted eventually), and we have to accommodate many million coefficients of possibly different magnitudes.

A	B	C	D	E	F	G	H	I
period	degree	seconds	hours	Iterations/d	Iter/d $\ln d$	Iter/d $\ln^2 d$	$\mu s/d \ln^2 d$	$\mu s/d \ln^3 d$
n	2^n							
12	4096	1	0.00	756	90.89	10.93	3.53	0.42
13	8192	3	0.00	798	88.56	9.83	4.51	0.50
14	16384	7	0.00	1053	108.51	11.18	4.54	0.47
15	32768	16	0.00	1220	117.34	11.29	4.52	0.43
16	65536	37	0.01	1399	126.15	11.37	4.59	0.41
17	131072	87	0.02	1585	134.51	11.42	4.78	0.41
18	262144	201	0.06	1786	143.15	11.47	4.93	0.39
19	524288	462	0.13	1988	150.95	11.46	5.08	0.39
20	1048576	1058	0.29	2210	159.42	11.50	5.25	0.38
21	2097152	2407	0.67	2437	167.42	11.50	5.42	0.37
22	4194304	5453	1.51	2678	175.62	11.52	5.59	0.37
23	8388608	18520	5.14	2945	184.73	11.59	8.69	0.54
24	16777216	32401	9.00	3204	192.60	11.58	6.98	0.42
25	33554432	34500	9.58	3457	199.50	11.51	3.42	0.20
26	67108864	76698	21.31	3738	207.42	11.51	3.52	0.20
27	134217728	320567	89.05	4044	216.08	11.55	6.82	0.36

FIGURE 8. Finding all periodic points of $p_2(z) = z^2 + 2$ for periods $n \leq 27$. The columns are similar to those in Figure 4. The observed complexity in terms of number of iterations is better than $O(d \ln^2 d)$ (see Figure 9), and in terms of computing time approximately $O(d \ln^3 d)$ (with some fluctuations; see Figure 10). The refinement threshold is $R = 0.05$, all roots were found.

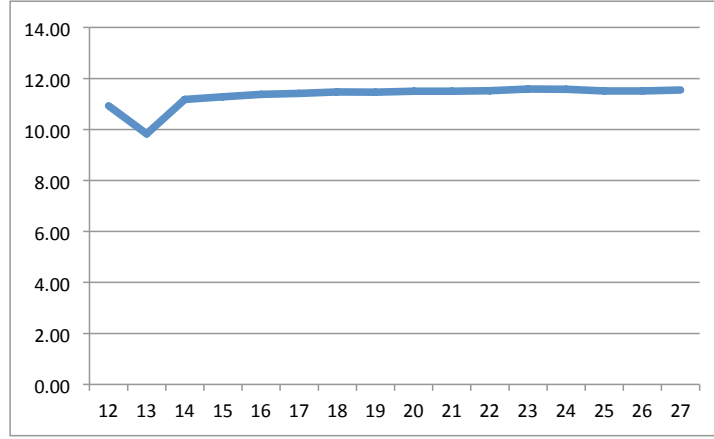


FIGURE 9. Number of iterations required for finding all periodic points of $z^2 + 2$ for periods $n \leq 27$ (divided by $d \ln^2 d$; column G in Figure 8). The complexity in terms of number of iterations seems to scale with $d \ln^2 d$ for periods $n \leq 27$, i.e. degrees up to 2^{27} (greater than 134 million).

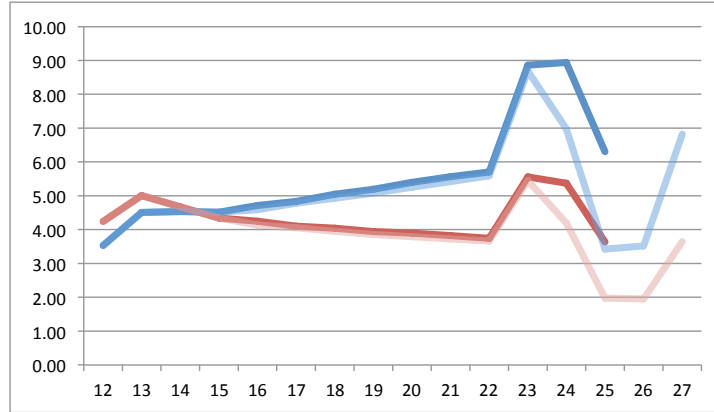


FIGURE 10. Iteration time required for finding all periodic points of $z^2 + 2$ for periods $n \leq 27$, divided by two scale functions. Blue curve: time divided by $d \ln^2 d$ (column H in Figure 8); red curve: time divided by $d \ln^3 d$ (column I), multiplied by 10 to adjust the scale. The time complexity seems to scale around $0.4 d \ln^3 d$ (in μsec , with some fluctuations). Two different runs of the same experiment are shown in different shades of color (the number of iterations remained the same).

4.3. Periodic points of $z^2 + i$. We performed the same experiments for $p_i(z) = z^2 + i$. The outcome was similar. We found (almost) all periodic points for periods up to $n = 27$, again with refinement threshold $R = 0.05$. This time, the computation was even faster: for period 27, it took 19 hours (but 3 roots remained missing); for $n = 26$, all roots were found in 9 hours.

A	B	C	D	E	F	G	H	I	J	K
period	degree	seconds	hours	Iterations/d	iter/d $\ln d$	iter/d $\ln^2 d$	$\mu s/d \ln d$	$\mu s/d \ln^{1.5} d$	$\mu s/d \ln^2 d$	roots missing
n	2^n									
12	4096	1	0.00	318	38.23	4.60	29.35	10.18	3.53	0
13	8192	1	0.00	351	38.95	4.32	13.55	4.51	1.50	0
14	16384	3	0.00	385	39.67	4.09	18.87	6.06	1.94	0
15	32768	7	0.00	418	40.20	3.87	20.55	6.37	1.98	0
16	65536	15	0.00	451	40.67	3.67	20.64	6.20	1.86	0
17	131072	33	0.01	485	41.16	3.49	21.37	6.22	1.81	0
18	262144	71	0.02	518	41.52	3.33	21.71	6.15	1.74	0
19	524288	153	0.04	551	41.84	3.18	22.16	6.11	1.68	0
20	1048576	332	0.09	585	42.20	3.04	22.84	6.13	1.65	0
21	2097152	716	0.20	618	42.46	2.92	23.46	6.15	1.61	0
22	4194304	1541	0.43	652	42.76	2.80	24.09	6.17	1.58	0
23	8388608	3309	0.92	685	42.97	2.70	24.74	6.20	1.55	0
24	16777216	7091	1.97	718	43.16	2.59	25.41	6.23	1.53	0
25	33554432	15139	4.21	752	43.40	2.50	26.04	6.25	1.50	0
26	67108864	32325	8.98	785	43.56	2.42	26.73	6.30	1.48	0
27	134217728	69302	19.25	818	43.71	2.34	27.59	6.38	1.47	3

FIGURE 11. Finding all periodic points of $p_i(z) = z^2 + i$ of periods $n \leq 27$. The columns are similar to those in Figure 4. However, the time complexity scales better: Columns H, I, and J show that it scales essentially with $d \ln^{1.5} d$ (see Figure 13). The complexity in terms of iterations (Columns F and G) scales with $d \ln d$ (Figure 12). The refinement threshold is again $R = 0.05$. The last column specifies the number of roots missed: for $n = 27$, exactly three roots were missed.

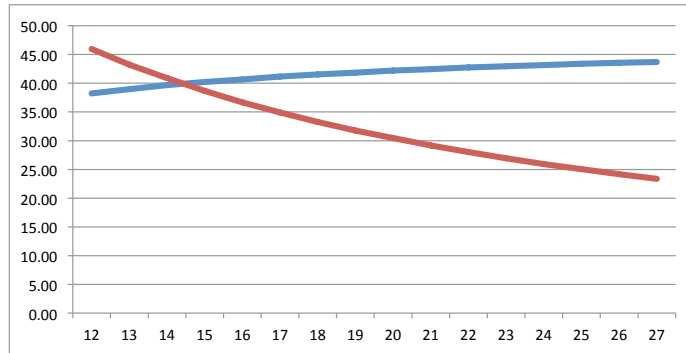


FIGURE 12. Observed complexity for finding all periodic points of $z^2 + i$ for periods $n \leq 27$. Blue: iterations/d $\ln d$ (column F in Figure 11); red: iterations/d $\ln^2 d$ (column G), multiplied by 10. The number of iterations seems to grow slightly faster than $4d \ln d$, but much slower than $4d \ln^2 d$.

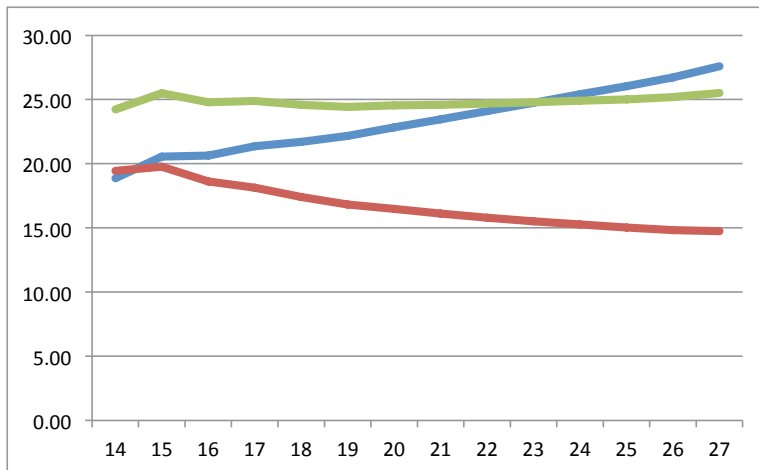


FIGURE 13. Iteration time (in μsec) required for finding all periodic points of $z^2 + i$ for periods $n \leq 27$, divided by three scale functions. Blue curve: time divided by $d \ln d$ (column H in Figure 11); red curve: time divided by $d \ln^2 d$ (column J), multiplied by 30 to adjust the scale; green curve: time divided by $d \ln^{1.5} d$ (column I), multiplied by 4. The time complexity seems to scale with $6.3 d \ln^{1.5} d$.

One might argue that 3 out of 134 million roots missing are negligible. However, our goal is to find all roots without exception. Of course we could have repeated the last experiment with slightly improved sensitivity. Instead, we recovered the missing roots by different methods; see Section 6). Interestingly, one was very near $z = 0$ and the other two were very near $z = i$.

4.4. Computing time per iteration. The computing time seems to oscillate somewhat unexpectedly in some of our experiments (see for instance Figure 10). A natural explanation might come from the fact that the computers had some other tasks running on separate threads (perhaps on separate cores), and while the actual computations would probably require the same amount of computing time, our memory-intense computations might suffer from unexpected memory swaps to the disk. A re-run of the most suspicious experiments showed of course the same number of Newton iterations (this part is deterministic), but also a rather similar oscillation of computing time as before (also shown in Figure 10).

A first step to analyze this behavior consists of comparing computing time per number of Newton iteration steps. Since every Newton iteration for our degree d polynomials requires complexity $O(\ln d)$, we show in Figure 14 computing time divided by $\ln d$. Not surprisingly, this is roughly constant for most series of experiments. Interestingly, for the periodic points of $z^2 + i$ one observes a certain decrease in d . This might be explained by the fact that not most of the computing time is spent on the Newton iterations, but other tasks require a significant fraction of the time.

Of course, all computing times specified depend on the computer and its software used, in particular efficiency of the compiler and the number of cores available.

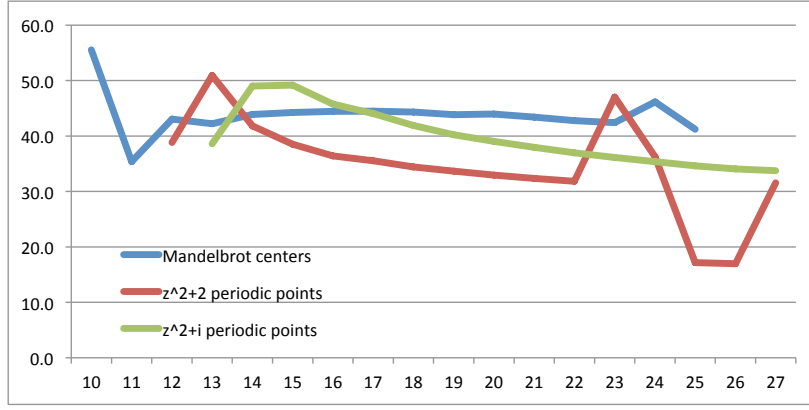


FIGURE 14. Computing time in nanoseconds per iteration, divided by $\ln d$ because every evaluation of the polynomial should roughly have complexity $\ln d$; combined for the three sets of experiments. Interestingly, for some experiment these numbers seem to decrease. (This seems to indicate that the Newton iterations do not use most of the computing time, perhaps because the recursion can be implemented particularly efficiently, so other tasks use a definite proportion of the computing time.) Of course, absolute numbers depend on the specific hard- and software used, but the relative behavior should be of structural interest.

Therefore, the absolute scales are machine dependent and not very relevant; however, we believe that the relative behavior for changing degrees is of structural interest. The issue of computing time oscillations warrants further study (but stays within a factor of 2, often less, and is thus not relevant to the key findings of our experiments).

5. GUARANTEEING THAT ALL ROOTS ARE FOUND

5.1. Guaranteeing that all roots are found. We admit that our method is heuristic: we do not have an a-priori guarantee that all roots will be found by the Iterated Refinement Newton Algorithm. However, there are several possibilities for testing a posteriori that all roots have indeed been found. Several of them have been described in detail in [SSt1, Sections 2.3 and 2.4], so we only briefly mention them here; an additional method based on the Newton identities is described in more detail.

An easy observation shows that for any polynomial p of degree d with associated Newton method $N_p(z) = z - p(z)/p'(z)$, and for arbitrary $z \in \mathbb{C}$, at least one root is contained in the disk $D_s(z)$ with $s = d \cdot |N_p(z) - z|$: so small Newton displacement steps $N_p(z) - z$ occur only near the roots. The factor of d can be significantly improved if the approximate locations of many roots are already known [SSt1, Lemma 4]. Therefore, if we have d points (coming from d different Newton orbits) that all come with small disjoint disks containing at least one root, then they together describe all roots. This method worked in practice for all degree 2^{20} polynomials investigated in [SSt1], but it requires that there are no near-multiple roots.

An independent and simpler test can be performed based on the Viète formulas: the coefficients of the polynomial are the elementary symmetric functions of the roots and encode for instance the sum of all roots or their product, and thus provide independent tests of success. More systematically, one can easily compute the value of the power sums $a_k := \sum_i \alpha_i^k$ over all roots α_i directly from the polynomial p ; this approach is based on the well-known Newton identities that relate the power sums to the coefficients of p . Comparing these with the power sums of all roots found is a natural generalization of the Viète tests. We describe this in detail below; the overall results are as follows.

In all experiments we describe, we found all roots according to the stopping and distinction criteria described in Section 3, and then used the power sum criterion to verify the roots found. For all our polynomials, even of maximal degrees, the power sum tests for exponents x^1 until x^{19} were run successfully and give reason to conclude that Newton's method with our ad-hoc stopping criterion described above (requiring that $|N_p(z) - z| < \varepsilon_{\text{stop}}$) found all roots with a typical accuracy of no more than $3 \cdot 10^{-16}$.

As an example, for the $d = 2^{27}$ periodic points of $z \mapsto z^2 + 2$ of period 27, the numerically computed power sum for x^1 was $-1.388 \cdot 10^{-12} + 2.220 \cdot 10^{-15}i$ with a deviation of less than $1.4 \cdot 10^{-12}$ from the true integer value 0 computed via the Newton identities. If we assume that all roots were computed with equally and independently distributed errors of expected size δ , so that the summed error can be viewed as a Brownian motion, then the total error of the sum should be on the order of $\delta \cdot \sqrt{d}$. We can thus estimate $\delta \approx 1.4 \cdot 10^{-12} \cdot 2^{-13.5} \approx 1.2 \cdot 10^{-16}$. These results are discussed in Section 5.3.

5.2. Newton identities. Here we describe a systematic method for checking whether indeed all roots have been found that is based on the well known Newton identities (also known as Newton-Girard formulas) that relate the coefficients of a polynomial (which are up to sign the elementary symmetric functions of its roots) to the power sums of all the roots. This method also makes it possible to locate missing roots in case not all were found (see Section 6).

The idea is to use these identities to compute from the coefficients of the polynomial p the sums of the powers of all roots of p , then to subtract from these the sums of powers of all roots found: the closer this difference is to 0, the better the accuracy of the roots found.

Of course it is difficult to find all coefficients of p (and many of them often have extremely large absolute values); but in order to compute the first m power sums of the roots, only the top m coefficients are required (beyond the leading coefficient that we usually scale to 1 anyway). For large degrees, even the top few coefficients can become very large in practice, and this may present numerical challenges.

We start by writing $p(z) = \prod_{i=1}^d (z - \alpha_i) = \sum_{k=0}^d c_k z^{d-k}$. We have the leading coefficient $c_0 = 1$, and the subsequent coefficients c_k are (up to sign) the elementary symmetric polynomials in the roots

$$c_1 = -\sum_i \alpha_i, \quad c_2 = \sum_{i < j} \alpha_i \alpha_j, \quad c_3 = -\sum_{i < j < k} \alpha_i \alpha_j \alpha_k, \quad \text{etc.}$$

We need the power sums

$$a_k := \sum_{i=1}^d \alpha_i^k$$

for $k \leq m$. They can be computed from the coefficients c_k by the Newton identities as follows:

$$\begin{aligned} -a_1 &= c_1 \\ -a_2 &= c_1 a_1 + 2c_2 \\ -a_3 &= c_1 a_2 + c_2 a_1 + 3c_3 \\ -a_4 &= c_1 a_3 + c_2 a_2 + c_3 a_1 + 4c_4 \end{aligned}$$

and so on for subsequent coefficients. Therefore, each a_k can be computed recursively from the c_k and the previously computed a_i .

In our applications, the polynomial p is not given in coefficient form, so in the first step the coefficients c_k have to be computed. Only the coefficients $c_1 \dots c_m$ are required. They can be found easily using the recursion that defines p : if in the recursion in every step only the m top-most coefficients are kept, then the final result will correctly yield the desired coefficients c_1, \dots, c_m .

To compute the coefficients c_k , one needs $\ln d$ recursion steps, each with complexity m^2 (or less if more efficient multiplication is implemented), for a total of $O(m^2 \ln d)$. The theoretical power sums a_k for $k \leq m$ are computed through the Newton identities, which are a triangular system of linear equations of dimension m , so they are evaluated in complexity $O(m^2)$. Both steps are negligible compared to the computation of the actual power sums from the roots found, which has complexity $O(dm)$.

5.3. Newton identities in practice and precision estimates. For the three families of polynomials, we computed the power sums for exponents $k \leq 19$, and compared them to the actual values computed using the Newton identities. The differences are shown in Figure 15.

The observed errors are between 10^{-15} and 10^{-8} , not surprisingly with increasing errors for higher degrees and higher powers. Within reasonably expected numerical error (discussed below), we see these computations as convincing confirmation that indeed all roots were found.

One way to model the error in these tests is to assume that all roots found have numerical errors of average size δ , distributed equally and independently. Specifically for the power sums with exponent $k = 1$, the sum is then a random walk of d steps with average step size δ , so the accumulated error should be of size $\delta\sqrt{d}$. Figure 16 shows that the results for $k = 1$ are consistent with the interpretation that typically $\delta < 3 \cdot 10^{-16}$.

For $k > 1$, we encounter a precision issue for large numbers. Let ρ be the largest absolute values of the roots of a given polynomial; we have $\rho \approx 2$ for the Mandelbrot centers, $\rho \approx 1.74$ for periodic points of $z^2 + 2$, and $\rho \approx 1.48$ for periodic points of $z^2 + i$. Then $|\alpha_i^k| \approx \rho^k$ for some i . These numbers increase exponentially with large k and are eventually subtracted from each other, or from the exact coefficients that in our cases are (Gaussian) integers, so (for number formats with fixed lengths mantissae) the absolute errors in the sums increase with ρ^k ; this would lead to approximately linear graphs in the accuracies shown in Figure 15, with slope $\ln \rho$. Indeed, the average slopes are close to 2.26 (for the centers of the Mandelbrot set), to 1.92 (for periodic points of $z^2 + 2$), and to 1.65 (for periodic points of $z^2 + i$); so in all three cases they are comparable to, and somewhat larger, than the predicted

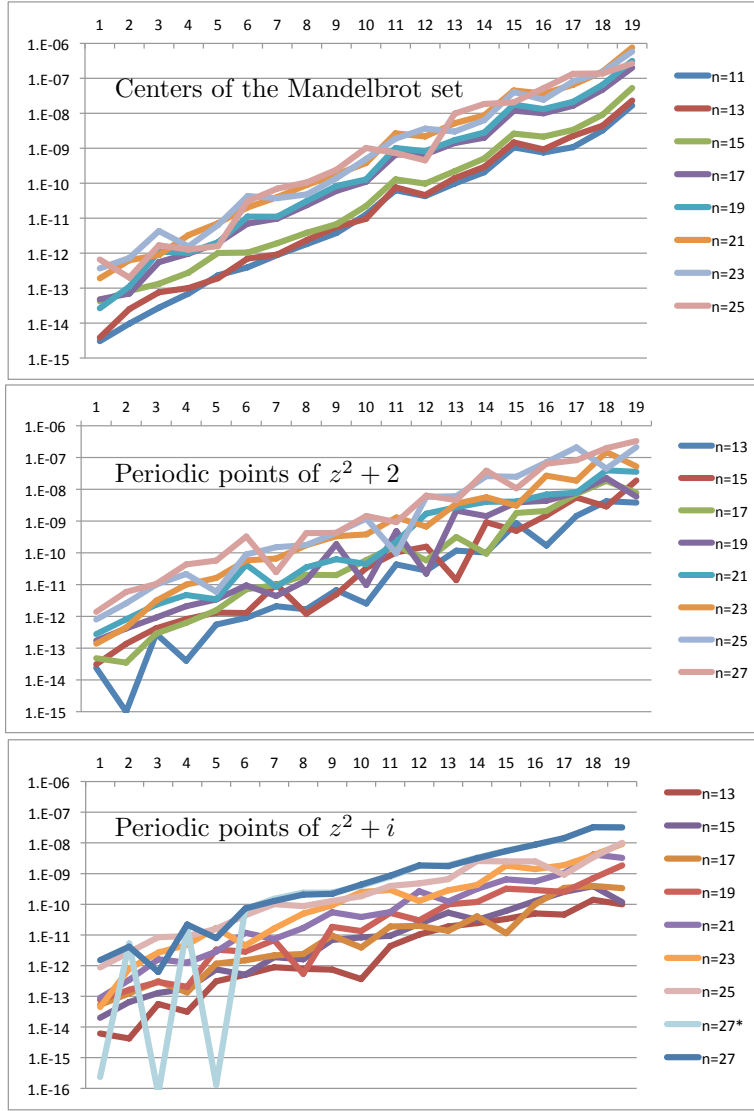


FIGURE 15. A posteriori verification that all roots were found, using the Newton identities for exponents $k \leq 19$ (horizontal scale), for odd periods n . Top: centers of the Mandelbrot set; middle: periodic points of $z^2 + 2$; bottom: periodic points of $z^2 + i$. Note that for the latter polynomials, the period $n = 27^*$ graph includes three roots that were recovered using the Newton identities. Therefore, the values for $k = 1, 3, 5$ are especially small (these were used to reconstruct the three roots), while the others are relatively large (due to the fact that the reconstructed roots have less precision than the others). The final $n = 27$ graph is based on all roots found with maximal precision (see Section 6.4).

lower bounds of the accuracy. In any case, the most accurate tests for the achieved accuracy are those with low exponents k .

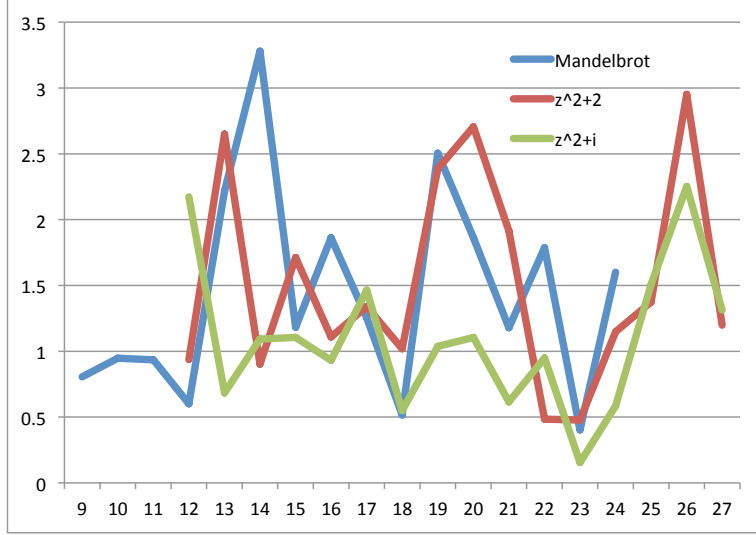


FIGURE 16. Expected precision of all roots, in units of 10^{-16} , for the three series of experiments (vertical). The horizontal scale shows N so that the degree is 2^N (the periodic points have period N , and the Mandelbrot centers have period $N+1$). Observe that all values are of approximately constant size, mostly within a factor of 5. These data are consistent with the interpretation that all roots have been found with typical accuracy of at most $\delta \approx 3 \cdot 10^{-16}$.

It might be interesting to report a few new challenges for these computations, coming from finite accuracy. One can use the Newton identities in both directions: from the numerically computed power sums one can compute the coefficients and compare these with the true values from the polynomials, or do the computation backwards starting with the coefficients and compare the power sums. The latter approach, from coefficients to power sums, seems more natural in our case because we do most computations with (Gaussian) integers, so without losing precisions. However, this involves *much* larger numbers (our polynomials have very large coefficients, and we had to use integer arithmetic that can handle products of these without losing valid digits); the computation in the other direction does not have the advantage of integers, but the numbers required are significantly smaller.

Another observation is that the very summation of the root powers loses valid digits. We had mentioned above that especially the high powers of the roots have limited absolute precision. In addition, when adding these large numbers, we often encounter very large sums (consistent with the large predicted values from the Newton identities). For instance, the powers $k = 18$ for period $n = 27$ of periodic points of $z^2 + 2$ sum up to about $-2 \cdot 10^{11}$, which reduces the available absolute accuracy further. Even when the power sum has small absolute value (such as for $k = 19$ for the same polynomial), the intermediate sums might become large before decreasing eventually (especially when the roots found are sorted for instance by

increasing real parts). We thus ran the addition a second time modulo 1 and modulo i , hence keeping track only of the fractional parts of the sums (having checked that the integer part of the solution is consistent), so that all available accuracy can measure the difference to the exact integer value.

5.4. Conclusion. The overall outcome of our three sets of experiments is that it is possible to find all roots of univariate polynomials of very large degrees in near-optimal complexity, both in terms of required Newton iterations and in terms of computing time. All roots could be located for degrees up to 134 million, and the complexity per root is constant up to small logarithmic factors. The accuracy of all roots found is better than $3 \cdot 10^{-16}$, comparable to the stopping condition $|z_n - z_{n+1}| < \varepsilon_{\text{stop}} = 10^{-15}$.

In one of the experiments, for periodic points of period $n = 27$ of $z^2 + i$, three of the more than 134 million roots were missed. These could be recovered with good precision by a method described in Section 6.

6. FINDING MISSING ROOTS

A structural disadvantage of our iterated refinement Newton root-finding method is that it does not come with an a priori guarantee that it will find all roots, and if in the end it turns out that some are missing, it is not clear where a refinement of starting points will recover the missing roots. This is different from the simpler linear scheme described in [SSt1]: there we usually start with $4d$ equidistributed points on a single circle, and if some roots are missing, then this initial set of points can get refined for additional orbits. The crucial difference is that in the linear scheme from [SSt1], every orbit starts at the same large circle surrounding all roots, so every orbit needs to run $O(d)$ iterations before even getting close to the roots. In our Iterated Refinement Newton Algorithm, most orbits start very close to the roots, so far fewer iterations are required, but the refinement takes place away from the locus where we have good control. We gain a lot of speed at the expense of giving up guaranteed convergence.

Indeed, in some experiments it does happen that the Iterated Refinement Newton Algorithm finds almost all roots, but several of the more than one million roots are missing (in one case that we describe in detail, three out of 134 million roots were missing). Roots may be missing either because the refinement was not sensitive enough, or because the roots were found but not declared sufficiently different from nearby roots that were reported.

Explicit deflation (dividing the original polynomial by the product of all linear factors corresponding to the roots found) is tempting but not an option: just building all coefficients from the roots found has quadratic complexity, so this involves much higher effort than all of root finding. Our global scheme does not have a way to pinpoint where the missing roots were lost, and thus where additional starting points are required to spot these.

Here we describe three methods, presumably well known, that have low complexity and that find missing roots efficiently when the number of remaining roots is small: if the given polynomial $p(z) = \prod_{i=1}^d (z - \alpha_i)$ has degree d and m roots are missing, then the complexity of this post-processing is essentially $O(dm)$, up to logarithmic factors. We are grateful to Dario Bini, Victor Pan and Michael Stoll for helpful suggestions here, and to Marvin Randig for substantial computational contributions.

6.1. Implicit deflation. The first method is “implicit deflation”. Suppose the roots found are $\alpha_1, \dots, \alpha_{d-m}$ and the missing roots are $\alpha_{d-m+1}, \dots, \alpha_d$. Write $q(z) = \prod_{i=d-m+1}^d (z - \alpha_i)$. Of course we do not know q directly.

The idea of implicit deflation is to evaluate Newton’s method for $q(z)$ in terms of p and the roots found: we have $q(z) = p(z) / \prod_{i=1}^{d-m} (z - \alpha_i)$ and thus

$$(2) \quad \frac{q'(z)}{q(z)} = \frac{p'(z)}{p(z)} - \sum_{i=1}^{d-m} \frac{1}{z - \alpha_i}$$

(for products of polynomials, the logarithmic derivative is additive). We can thus evaluate $N_q(z) = z - q(z)/q'(z)$ using only $p'(z)/p(z)$ (which is computed by the usual Newton method for N_p) and the roots found so far.

If $m \ll d$, then the final sum dominates the complexity: every iteration step of N_q has complexity essentially $O(d)$, while the number of necessary Newton iterations depends only on the degree m of q . Depending on the efficiency of root finding for this (relatively) low degree polynomial, the number of required Newton iterations is between $O(m \ln m)$ and $O(m^2)$, so the total complexity of finding the remaining roots is between $O(md \ln m)$ and $O(m^2 d)$.

6.2. Ehrlich-Aberth iteration. A second method to locate the missing roots is the Ehrlich-Aberth method (which is underlying the successful implementation `MPSolve` to find all roots). This method is a parallel iteration in d variables, where the j -th component is the Newton method of $p(z) / \prod_{i \neq j} (z - \tilde{\alpha}_i)$ and the $\tilde{\alpha}_i$ are the current approximations to all d roots. As with implicit differentiation, one does not need to compute $\prod_{i \neq j} (z - \tilde{\alpha}_i)$ explicitly for the Newton displacement, but uses an analog of (2).

All d coordinates are updated simultaneously. If $d - m$ roots are already known, then this method is an iteration in only m variables. In particular, if $m = 1$, then implicit deflation equals the Ehrlich-Aberth method with $d - 1$ coordinates fixed; these two methods are thus closely related.

6.3. Newton identities. A third method for locating missing roots is based on the Newton identities described in Section 5.2 that relate the coefficients of a polynomial (which are up to sign the elementary symmetric functions of its roots) to the power sums of all the roots. This method may be in less frequent use, even though it is based on very classical ideas. It allows one to construct a polynomial q of degree m the roots of which are exactly the missing roots of p . This polynomial will be constructed in terms of its coefficients using sums of different powers of the roots as well as Newton’s identities.

The idea is to use these identities to compute from the coefficients of the polynomial p the sums of the powers of all roots of p , then to subtract from these the sums of powers of all roots found: we thus obtain the sums of powers of the missing roots, and from these one can compute the coefficients of q by applying the Newton identities backwards (where q as above is the polynomial that has exactly the missing roots). We describe this method in somewhat more detail.

We described in Section 5.2 linear relations that allow one to compute from the coefficients of the polynomial p the sums a_k of the k -th powers of all its roots.

Now suppose the $d - m$ roots $\alpha_1 \dots, \alpha_{d-m}$ of p have already been computed, and m roots $\alpha_{d-m+1}, \dots, \alpha_d$ are missing. We can compute their power sums

$$b_k = \sum_{i=d-m+1}^d \alpha_i^k = \sum_{i=1}^d \alpha_i^k - \sum_{i=1}^{d-m} \alpha_i^k = a_k - \sum_{i=1}^{d-m} \alpha_i^k$$

from the a_k and the $d - m$ roots found. Finally, we are looking for the polynomial

$$q(z) = \prod_{i=d-m+1}^d (z - \alpha_i) = \sum_{k=0}^m e_k z^{m-k},$$

where the e_k are the coefficients of q . The coefficients e_k can be determined from the b_j using the Newton identities backwards:

$$\begin{aligned} -e_1 &= b_1 \\ -2e_2 &= e_1 b_1 + b_2 \\ -3e_3 &= e_2 b_1 + e_1 b_2 + b_3 \\ -4e_4 &= e_3 b_1 + e_2 b_2 + e_1 b_3 + b_4 \\ &\vdots \end{aligned}$$

Therefore, once we know b_1, \dots, b_m , we can find q in coefficient form and then apply any root finder to this polynomial of degree m .

In order to find the m coefficients e_1, \dots, e_m of q , only the power sums b_1, \dots, b_m are required in the Newton identities backwards, and hence only the power sums a_1, \dots, a_m and the coefficients c_1, \dots, c_m are required. We described earlier in Section 5.2 that these coefficients can be computed with little effort even when our polynomials p are not given in coefficient form.

The complexity of this method has been described in Section 5.2: it is dominated by the computation of the power sums $b_1 \dots b_m$, which requires $O(dm)$ operations. The remaining computations for m missing roots can be performed in $O(m^2 \ln d)$ operations.

One structural advantage of this method is that the original roots are required only for the simple computation of the power sums, not for the subsequent root finding iteration. In our applications, the roots were found on a different computer system than the one on which the missing roots were located, so not all $d - m$ previously found roots had to be transmitted, but only m easily computed power sums (it makes a substantial difference to transmit, or even to store, $2^{27} - 3$ complex roots or 3 power sums). This advantage comes with a certain disadvantage, though: the inaccuracies of all $d - m$ roots contribute equally to inaccuracies in the power sums, no matter how far they are from the missing roots; in practice this limits the achievable accuracy of this method (for the methods of implicit deflation or Ehrlich–Aberth, the inaccuracy of roots that are far from missing roots becomes far less important).

The two purposes of Newton identity, to find missing roots and to check the roots found and estimate the accuracy achieved, can be combined: after m missing roots have been reconstructed using the first m coefficients, one can compute m' additional coefficients and hence m' additional power sums, and use these to verify that now all roots have been found with great accuracy.

power	power sum of all roots		sums of powers of roots found (integer+fractional part)						accuracy achieved
			integer part		fractional part				
1	0				-2	-1.37057E-05	-4.5743E-06	2.3646E-16	
2	-134217728		2	-134217728		6.53912E-11	-2.3933E-09	5.23677E-12	
3	0				2	3.40316E-09	3.54708E-10	9.99201E-16	
4	-134217728	-134217728	-134217730	-134217728		-4.56786E-10	4.5536E-09	1.92751E-11	
5	0				-2	-5.67929E-09	-5.861E-10	1.3205E-16	
6	-402653184	134217728	-402653182	134217728		7.5283E-10	-6.8655E-09	6.9785E-11	
7	0				2	7.82265E-09	8.7303E-10	1.50924E-10	
8	671088640		-2	671088640		-1.01565E-09	8.88256E-09	2.3954E-10	
9	0				-2	-1.00335E-08	-9.3422E-10	2.38844E-10	
10	671088640	536870912	671088642	536870912		7.47176E-10	-1.1407E-08	3.99815E-10	
11	0				2	1.32488E-08	9.20673E-10	7.81338E-10	
	real	imag	real	imag		real	imag	abs value	

FIGURE 17. Finding three missing roots using the power sum method. The first column shows the powers, the second one the power sum of all roots (using the Newton identities, separated into real and imaginary parts), the third the power sum of all $2^{27} - 3$ roots found (separated into integer and fractional parts, and displaying less accuracy than used), and the last column shows the numerical accuracy achieved for the sum of all 2^{27} roots, including the 3 recovered roots and the predicted value from column 2. The odd powers 1, 3, 5 were used to reconstruct the three missing roots and thus show large computational accuracy. The remaining powers 7, 9, 11 indicate that all roots have been found with about ten digits accuracy (probably much better for most roots).

6.4. Newton identities in practice. In this section (performed in cooperation with Marvin Randig) we briefly describe how the reconstruction of the 3 missing roots among the 2^{27} periodic points of period 27 of $p_i(z) = z^2 + i$ works, and illustrate the results and some practical challenges. We are interested in the roots of $p(z) = p_i^{\circ n}(z) - z$ for $n = 27$. The coefficients are Gaussian integers, and the first few have the following values:

$$c_0 = 1, \quad c_1 = 0, \quad c_2 = 2^{26}i, \quad c_3 = 0, \quad c_4 = -2^{51} + 2^{25} + 2^{25}i, \quad c_5 = 0$$

with $2^{51} \approx 2.25 \cdot 10^{15}$; the subsequent c_k with k odd vanish, while those with k even are huge. The respective power sums from the Newton identities have comparable sizes. Subtracting the sum of 134 million numerically computed roots to obtain power sums of the three missing roots is a certain challenge to the numerics involved.

Nonetheless, in this case the three missing roots were found: two were close to i , and one was close to 0. This was done initially using the power sums 1, 2, and 3. However, we have $a_2 = -134217728i$, so in the subtraction 9 valid decimal digits are lost, which is a severe limitation to accuracy. Therefore, the computations were redone using the power sums 1, 3, and 5 for which no serious cancellations occur (the odd power sums of all roots should vanish). The results are shown in Figure 17. The third column shows that the odd power sums of the three missing roots must be equal to $\pm 2i$ with an accuracy of 10^{-5} , and indeed the three recovered missing

roots are approximately

$$\begin{aligned}
 (3) \quad & 0 \quad +13.7046 \cdot 10^{-6} + 4.57423 \cdot 10^{-6}i \\
 & i \quad -8.37799 \cdot 10^{-7} + 4.39432 \cdot 10^{-7}i \\
 & i \quad -8.36666 \cdot 10^{-7} - 4.39313 \cdot 10^{-6}i .
 \end{aligned}$$

The accuracy of these computations can be checked in the remaining power sums, as shown in the rightmost column in Figure 17. The odd powers 1, 3, 5 show only that the computations were self-consistent: the three roots were recovered so that these power sums are correct within a computational accuracy of about 16 digits.

The other power sums provide independent tests. The fact that they all match to about ten valid decimal digits can be taken as experimental confirmation that all roots were found to at least this precision (but there are problems when two missing roots are very close to each other; see below). We believe that the initial $2^{27} - 3$ roots were found with accuracy of at least $3 \cdot 10^{-16}$ as in all other computations, so that the power sums have accuracy of around $3 \cdot 10^{-16} \cdot 2^{27/2} \approx 3.5 \cdot 10^{-12}$, and the accuracy of the three recovered roots should be comparable to this latter value (much lower than the remaining roots because of the accumulated error in the large sums). This is consistent with the power sum for $k = 2$, and we argued above why power sums for higher powers have less accuracy even when the roots themselves have small errors. (For even powers, when the power sums were close to large Gaussian integers, we again performed the computations twice, as described earlier: once to see that the sums are close to the desired Gaussian integers, and once modulo Gaussian integers to achieve higher absolute accuracy.)

With these approximations, the three missing roots were found also by a refinement of the original Newton iteration; compare Figures 18 and 19. Improved approximations to these three roots are

$$\begin{aligned}
 (4) \quad & (13.70461018 + 4.57422958i) \cdot 10^{-6} ; \\
 & i \quad + (9.67674561 - 0.07238806i) \cdot 10^{-10} ; \\
 & i \quad + (1.66887081 + 1.25381459i) \cdot 10^{-10} ,
 \end{aligned}$$

now with the same experimental accuracy as all other roots.

A posteriori analysis reveals why the three roots were originally not found by our ad-hoc stopping criteria: for the first root, with our numerical accuracy we only achieve $|N(z) - z| \approx 1.33 \cdot 10^{-15} > \varepsilon_{\text{stop}} = 10^{-15}$: we did have Newton orbits that “wanted to” converge to the root, but the stopping criterion was too strict. The Newton dynamics in Figures 1 and 18 (left) illustrates this: both show the same detail of the Newton dynamics around this missing root, and both were computed with the algorithm described — except that in Figure 1 the requirement that $|N(z) - z| < \varepsilon_{\text{stop}} = 10^{-15}$ was relaxed to $2 \cdot 10^{-15}$. The black points in Figure 1 are points that do not converge to roots with the initial choice of $\varepsilon_{\text{stop}}$. Estimates using [SSt1, Lemma 4] prove that the improved approximation to the first root is indeed close to a root with error at most $7.330 \cdot 10^{-15}$ (probably better).

The other two roots (both near i) were missed because they were closer than $\varepsilon_{\text{root}} = 10^{-14}$ from two other roots: in the first case, there was another root found at distance $(4.98 + 6.82i) \cdot 10^{-15}$, and in the second case at distance $(5.68 - 5.39i) \cdot 10^{-15}$, both somewhat smaller than 10^{-14} in absolute value.

As we pointed out, the values of the constants in our algorithm were chosen in an ad-hoc fashion and obviously have to be adjusted for polynomials of extremely

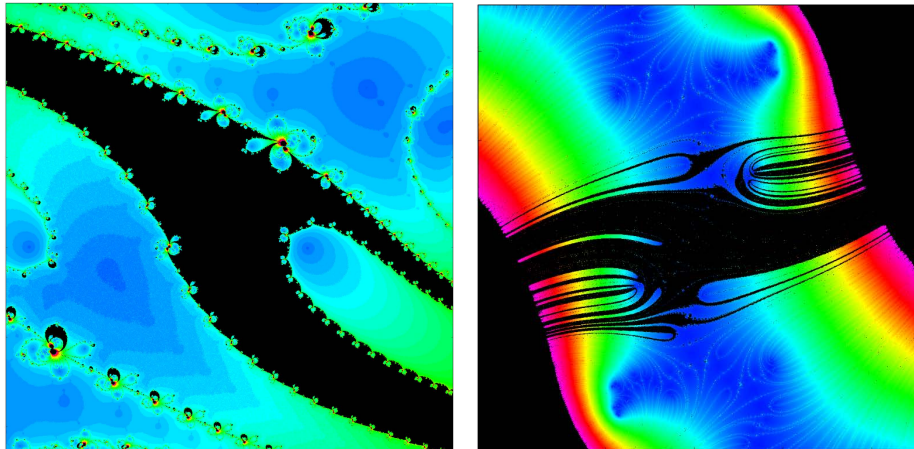


FIGURE 18. Newton dynamics for finding periodic points of period 27 for $z^2 + i$ (degree 2^{27}). Left: a detail near the missing root $1.37 \cdot 10^{-5} + 4.57 \cdot 10^{-6}i$. Black shows points for which the Newton method with $\varepsilon_{\text{stop}} = 10^{-15}$ failed to find a root. Comparison with Figure 1 (same detail for the same polynomial, with $\varepsilon_{\text{stop}} = 0.5 \cdot 10^{-15}$) shows that the problem was a sub-optimal stopping criterion: most black points actually converge to the missing root, which was however not recognized. Right: the same polynomial with an even finer constant $\varepsilon_{\text{stop}} = 10^{-16}$ (zoomed out): many more roots fail to be found. Pictures by Marvin Randig.

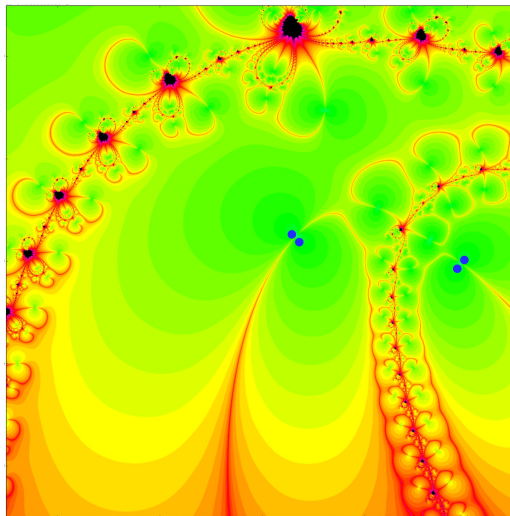


FIGURE 19. The Newton dynamics in a neighborhood of i . The two missing roots near i are indicated by blue dots together with the two nearby roots that were found (manually inserted, not to scale: the distance between the pairs is $8.1 \cdot 10^{-10}$, while the distance within both pairs is close to 10^{-14}). Picture by Marvin Randig.

large degrees — but the underlying Newton method is stable enough that it works well even with these ad-hoc criteria.

Once the three missing roots were found to full precision as in (4), we compared their values with the approximations from the Newton identities as in (3). For the root near $z = 0$, the error was $3.50745 \cdot 10^{-12}$, almost on the nose equal to the estimated value. However, for the two roots near i , the error turned out to be equal to $9.45195 \cdot 10^{-7}$ for both, much larger than expected. We provide an explanation below.

Observe first that the fact that both roots have errors with almost equal absolute values is due to the fact that they must match the sums of the first powers, so the deviations must be opposite to each other within the precision of the power sums.

The fact that the last two missing roots are very close to each other has to do with a particular property of our polynomial: all the roots we are looking for are periodic points of $p_i(z) = z^2 + i$, so as a set they are invariant by p_i . They are equidistributed with respect to harmonic measure on the Julia set of p_i , but all roots near $z = 0$ are contracted by the quadratic map p_i that has a critical point at $z = 0$, so the roots must be very close to each other near $p_i(0) = i$. The pairs of nearest roots are thus to be expected to be close to $z = i$, and the trouble was caused by two such pairs.

After the missing root near $z = 0$ is recovered numerically with expected precision, the two missing roots near $z = i$ form a quadratic equation with an (almost) double root, given with some precision ε . In such a case, the roots only have precision $\sqrt{\varepsilon}$, and this is what we experience here. It is an interesting fact that in the reconstruction of the last two missing roots, an extremely ill-conditioned quadratic equation causes substantial errors, while Newton’s method itself is able to find all 2^{27} roots (many of them much closer to each other than this pair), even the three initially missing roots and the nearby roots close to them, without any issues of ill-conditioning (except that we had to adjust our ad-hoc threshold parameters). Of course, ill-conditioning is an ubiquitous experience, but it is remarkable how well Newton can handle it.

REFERENCES

- [Ar] Ludwig Arnold, *Über die Nullstellenverteilung zufälliger Polynome* (On the distribution of roots of random polynomials). Math. Z. 92 (1966), 12–18.
- [AHU] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *The design and analysis of computer algorithms*, Addison Wesley, 1974.
- [BAS] Todor Bilarev, Magnus Aspenberg, and Dierk Schleicher, *On the speed of convergence of Newton’s method for complex polynomials*. Mathematics of Computation **85** (2016), 693–705.
- [BR14] Dario A. Bini and Leonardo Robol, *Solving secular and polynomial equations: a multiprecision algorithm*. Journal of Computational and Applied Mathematics **272** (2014), 276–292.
- [BLS] Béla Bollobás, Malte Lackmann, and Dierk Schleicher, *A small probabilistic universal set of starting points for finding roots of complex polynomials by Newton’s method*. Mathematics of Computation **82** (2013), 443–457.
- [DH] Adrien Douady, John Hubbard, *Etude dynamique des polynômes complexes I & II* (the “Orsay Notes”). Publ. Math. Orsay (1984–85).
- [ET] Paul Erdős, Pál Turán, *On the distribution of roots of polynomials*. Ann. of Math. (2) **51** (1950), 105–119.
- [F] Steve Fortune, *An iterated eigenvalue algorithm for approximating roots of univariate polynomials*, J. of Symbolic Computation **33** 5 (2002), 627–646.

- [HSS] John Hubbard, Dierk Schleicher, and Scott Sutherland, *How to find all roots of complex polynomials with Newton's method*. *Inventiones Mathematicae* **146** (2001), 1–33.
- [McN1] John M. McNamee, *A 2002 update of the supplementary bibliography on roots of polynomials*. *J. Comput. Appl. Math.* **142** 2 (2002), 433–434.
- [McN2] John M. McNamee, *Numerical methods for roots of polynomials. Part I. Studies in Computational Mathematics* **14**. Elsevier B. V., Amsterdam, 2007.
- [MB] Robert T. Moenck and Allan B. Borodin, *Fast modular transform via division*, Proc. 13th annual symposium on switching and automata theory 90–96, IEEE Comp. Society Press, Washington, DC, 1972.
- [P1] Victor Pan, *Approximating complex polynomial zeros: modified Weyl's quadtree construction and improved Newton's iteration*. *Journal of Complexity* **16** (2000), 213–264.
- [P2] Victor Pan, *Univariate polynomials: nearly optimal algorithms for factorization and rootfinding*, *Journal of Symbolic Computations* **33** 5 (2002), 701–733.
- [R] James Renegar, *On the worst-case arithmetic complexity of approximating zeros of polynomials*. *J. Complexity* **3** 2 (1987), 90–113.
- [S1] Dierk Schleicher, *On the number of iterations of Newton's method for complex polynomials*. *Ergodic Theory and Dynamical Systems* **22** (2002), 935–945.
- [S4] Dierk Schleicher, *On the efficient global dynamics of Newton's method for complex polynomials*. Manuscript (2013). arXiv:1108.5773.
- [SSch] Dierk Schleicher and Simon Schmitt, *Newton's method in practice III: a case study with ill-behaved polynomials of large degrees*. Manuscript, in preparation.
- [SSt1] Dierk Schleicher and Robin Stoll, *Newton's method in practice: finding all roots of polynomials of degree one million efficiently*. *Journal of Theoretical Computer Science*, to appear (2016). arXiv:1508.02935.